PROGRAMMING

FOR PROBLEM SOLVING

{ C LANGUAGE }

Module :- 9

Pointers

Start

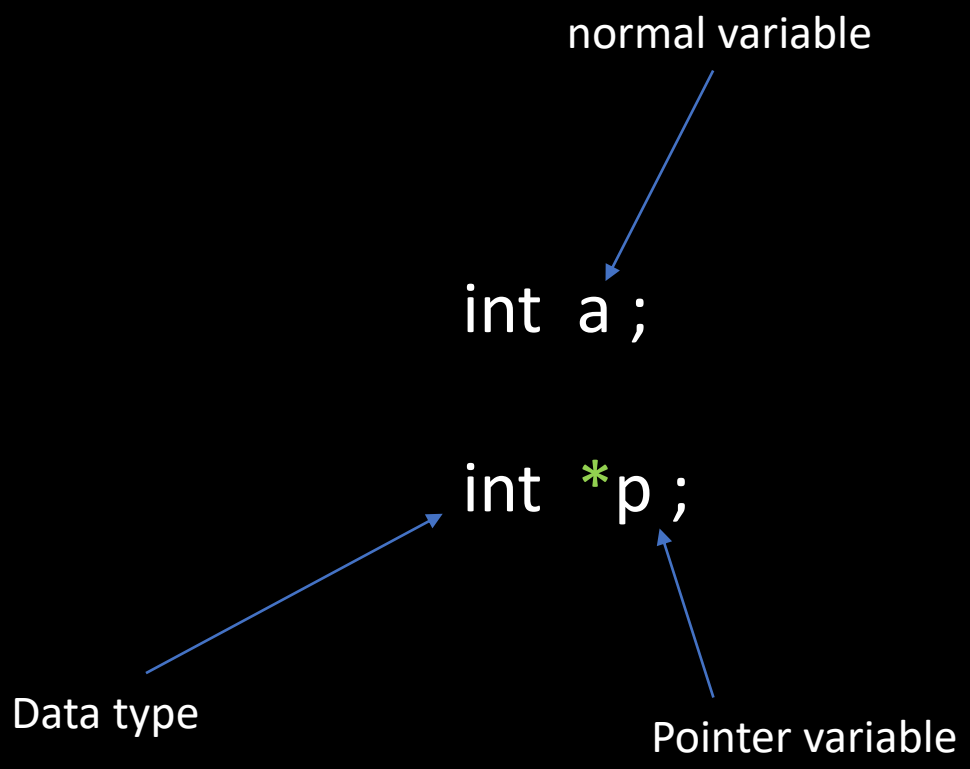
Module 9 :- Pointers

- Idea of pointers
- Defining pointers
- Use of pointers in self – referential structures
- Notion of linked list (no implementation)
- Pointer to pointer
- Pointer to array
- Pointer to string
- Array of pointer
- Pointer to function
- Pointer to structure

Pointer

- It is special type of variable which store the address of another variable
- It can store the address of same data type means an integer pointer can store the address of integer variable , character , pointer can store the address of character variable and so on
- If we add asterisk(*) symbol with any variable at the time of declaring variable the this variable is called pointer variable
- * symbol is used to get the value at address which is hold by pointer

Syntax



- Here a is normal variable
- P is pointer variable because it is associated with * symbol

How to Use Pointers?

The use of pointers in C can be divided into three steps:

1. Pointer Declaration
2. Pointer Initialization
3. Pointer Dereferencing

1. Pointer Declaration

In pointer declaration, we only declare the pointer but do not initialize it. To declare a pointer, we use the **(*) dereference operator** before its name.

```
int *ptr;
```

The pointer declared here will point to some random memory address as it is not initialized. Such pointers are called wild pointers.

2. Pointer Initialization

Pointer initialization is the process where we assign some initial value to the pointer variable. We generally use the **(&) addressof operator** to get the memory address of a variable and then store it in the pointer variable.

```
int var = 10;
int * ptr;
ptr = &var;
```

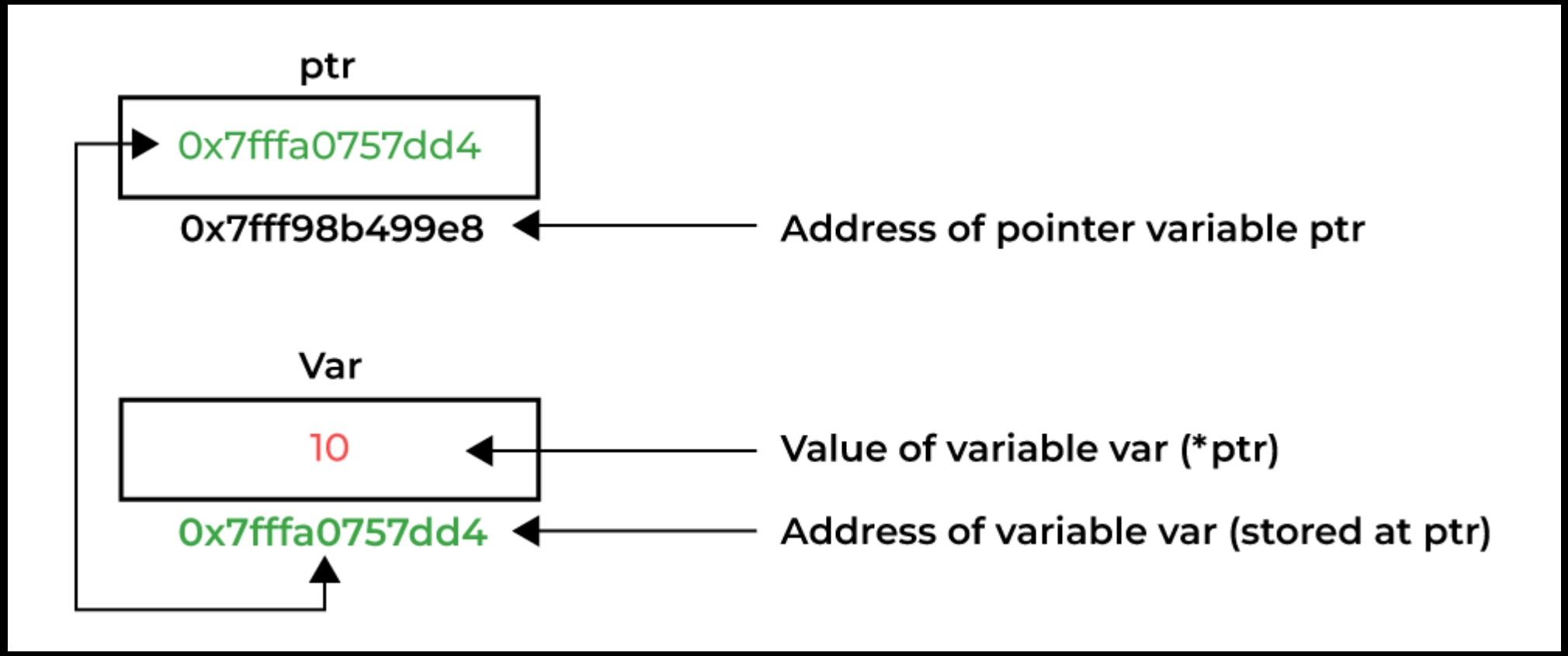
We can also declare and initialize the pointer in a single step. This method is called **pointer definition** as the pointer is declared and initialized at the same time.

```
int *ptr = &var;
```

Note: *It is recommended that the pointers should always be initialized to some value before starting using it. Otherwise, it may lead to number of errors.*

3. Pointer Dereferencing

Dereferencing a pointer is the process of accessing the value stored in the memory address specified in the pointer. We use the same **(*) dereferencing operator** that we used in the pointer declaration.



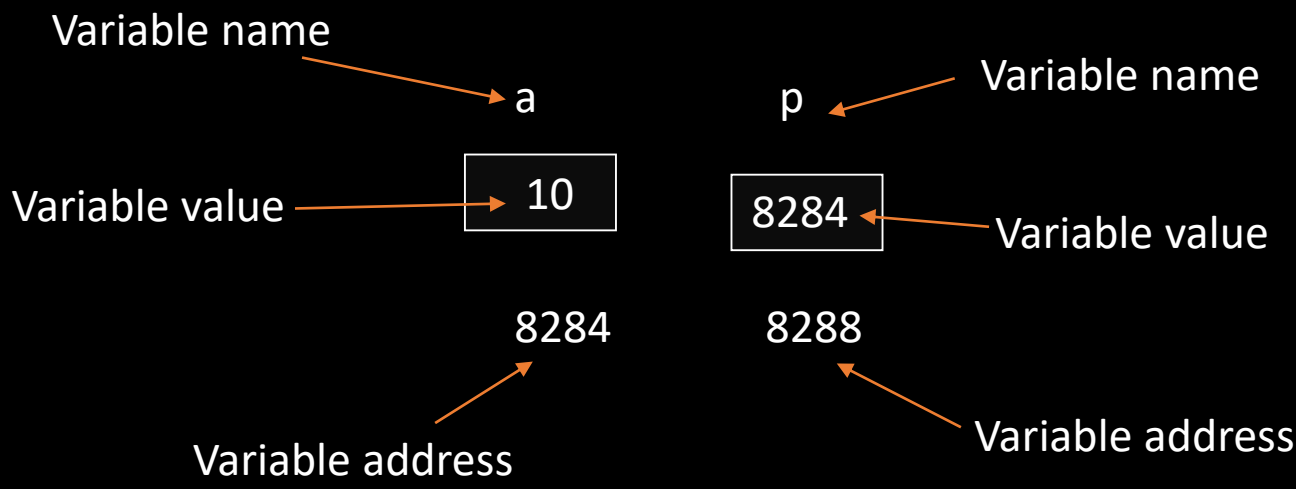
Example

```c
#include<stdio.h>
int main()
{
    int a=10;//initializing normal variable
    int *p;//declaring pointer variable
    p=&a;//address of variable a is assigned to p

    printf("value of a=%d\n",a);
    printf("address of a=%d\n",&a);
    printf("value of p=%d\n",p);
    printf("address of p=%d\n",&p);
    printf("value of *p=%d\n",*p);

    return 0;
}
```

Output
value of a=10
address of a=8284
value of p=8284
address of p=8288
value of *p=10

- Output explanation

Assume that the address of variable a is 8284 and address of variable p is 8288 it may be different in your system




```c
#include <stdio.h>

void kks()
{
    int var = 10;

    // declare pointer variable
    int* ptr;

    // note that data type of ptr and var must be same
    ptr = &var;

    // assign the address of a variable to a pointer
    printf("Value at ptr = %p \n", ptr);
    printf("Value at var = %d \n", var);
    printf("Value at *ptr = %d \n", *ptr);
}

int main()
{
    kks();
    return 0;
}
```

```
###Output###
Value at ptr = 0x7ffca84068dc
Value at var = 10
Value at *ptr = 10
```

Types of Pointers in C

There are majorly four types of pointers, they are:

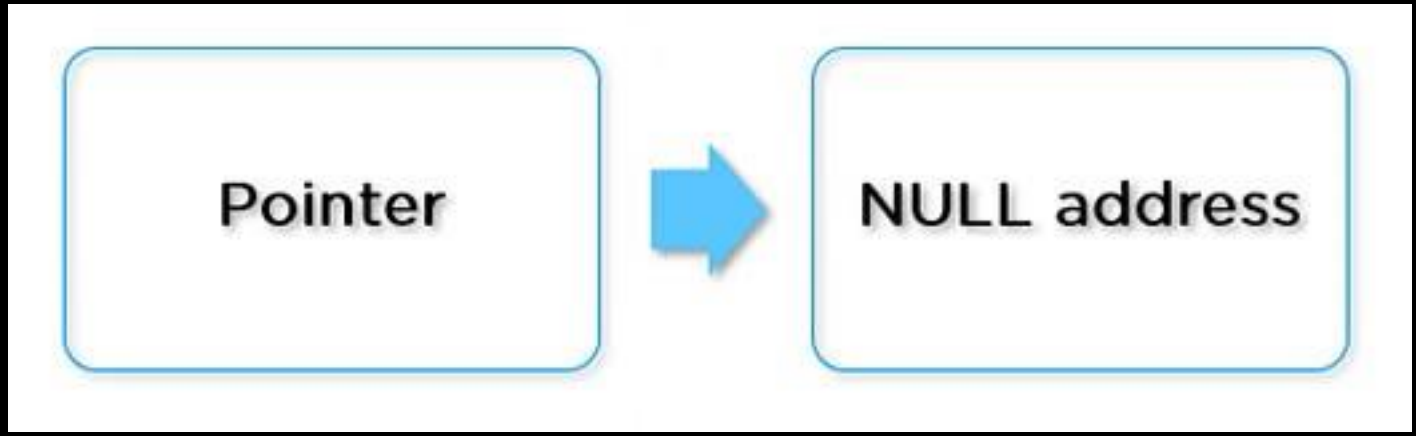
- Null Pointer
- Void Pointer
- Wild Pointer
- Dangling Pointer

Null Pointer:

If you assign a NULL value to a pointer during its declaration, it is called Null Pointer.

Syntax:

```
int *var = NULL;
```




```c
#include<stdio.h>

int main()
{
    int *var = NULL;

    printf("var=%d",*var);

}
```

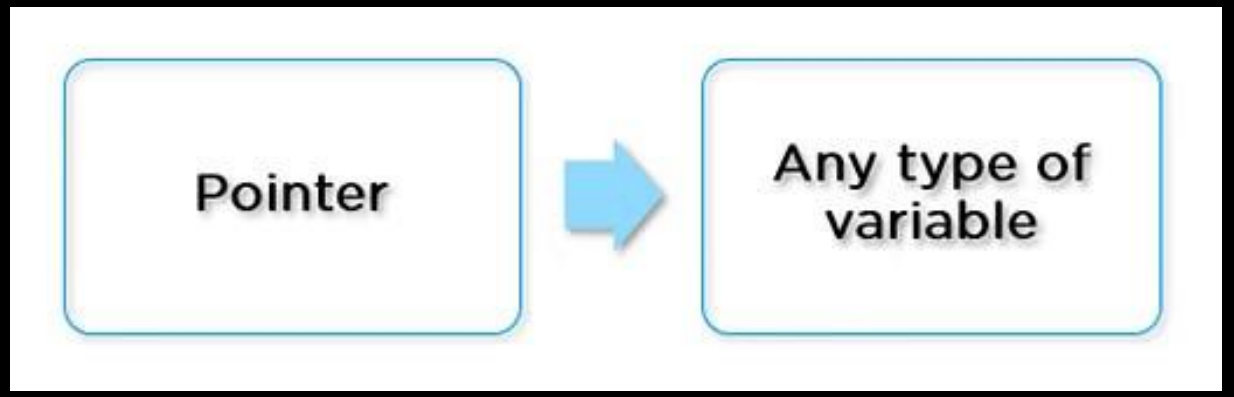
Void Pointer:

When a pointer is declared with a void keyword, then it is called a void pointer. To print the value of this pointer, you need to typecast it.

Syntax:

```
void *var;
```

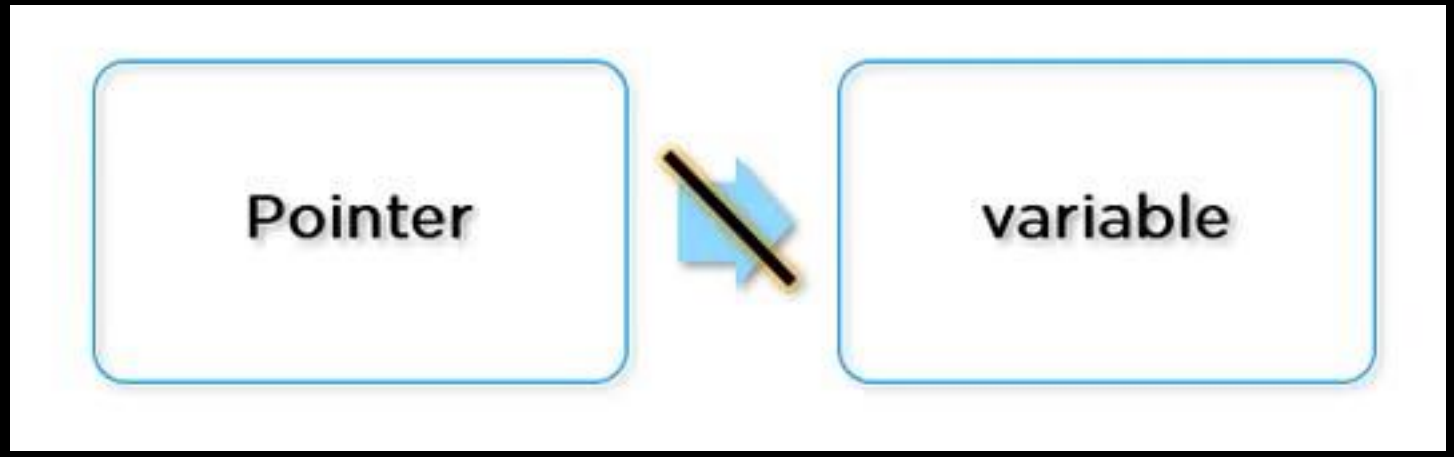
```c
#include<stdio.h>

int main()
{

    int a=2;
    void *ptr;

    ptr= &a;

    printf("After Typecasting, a = %d", *(int *)ptr);

    return 0;

}
```



Wild Pointer:

A wild pointer is only declared but not assigned an address of any variable. They are very tricky, and they may cause segmentation errors.

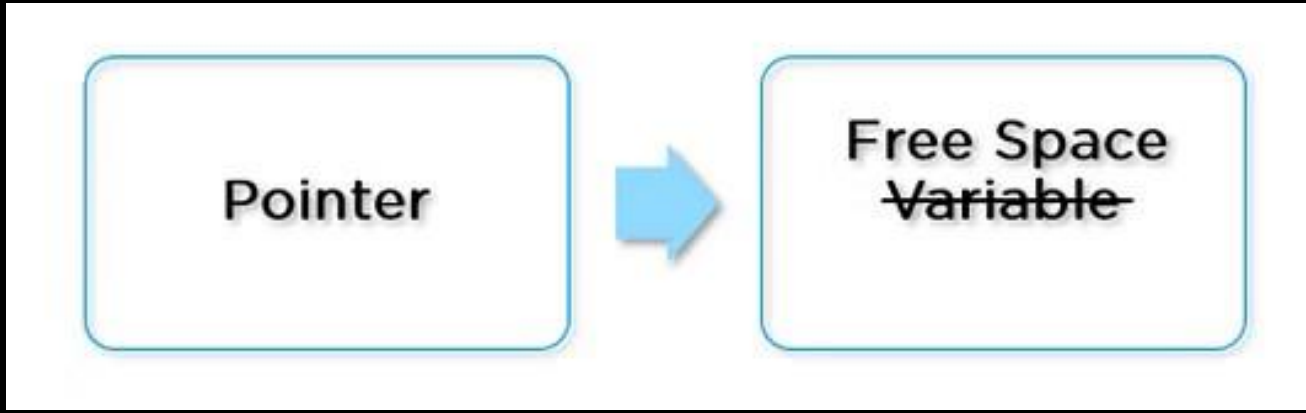
```c
#include<stdio.h>

int main()
{
  int *ptr;

  printf("ptr=%d",*ptr);

  return 0;

}
```



Dangling Pointer

•Suppose there is a pointer p pointing at a variable at memory 1004. If you deallocate this memory, then this p is called a dangling pointer.
•You can deallocate a memory using a free() function.

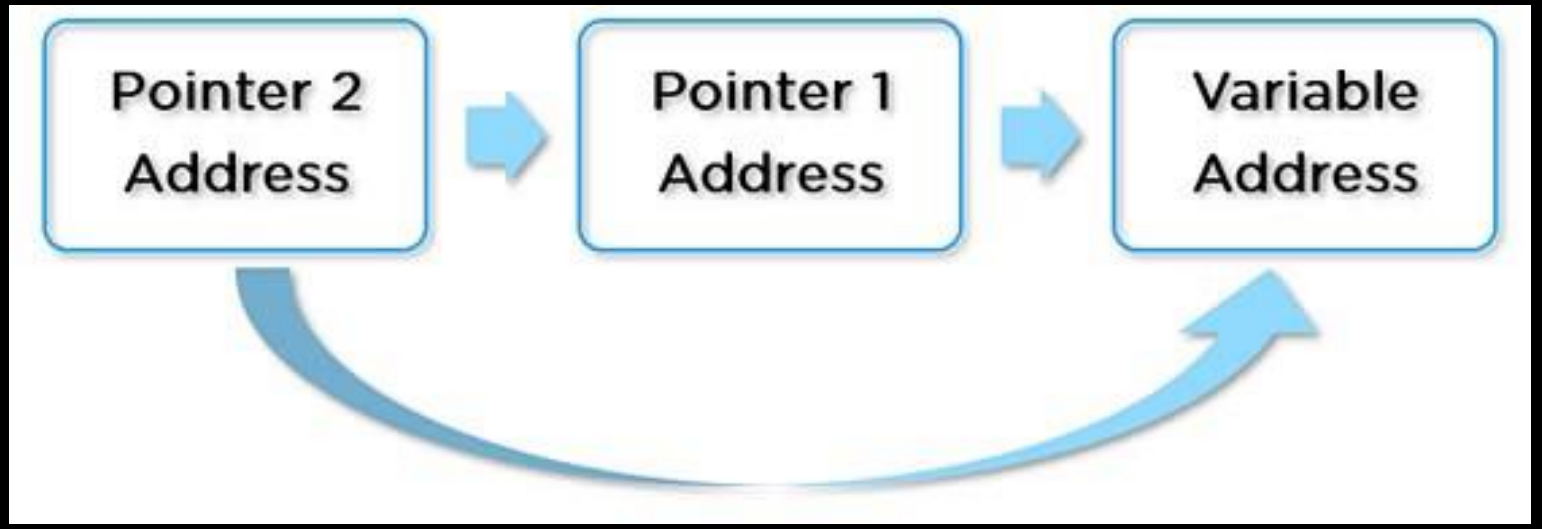
```c
#include<stdio.h>
#include<stdlib.h>

int main()
{

    int *ptr=(int *)malloc(sizeof(int));

    int a=5;
    ptr=&a;

    free(ptr);

    //now this ptr is known as dangling pointer.

    printf("After deallocating its memory *ptr=%d",*ptr);

  return 0;

}
```



Pointer to Pointer:

In this situation, a pointer will indirectly point to a variable via another pointer.

Syntax:
int **ptr;




```c
#include <stdio.h>

int main ()
{
    int var, *ptr1, **ptr2;
    var = 10;

    ptr1 = &var;
    ptr2 = &ptr1;

    printf("Value of var = %d\n", var );
    printf("Value available at *ptr1 = %d\n", *ptr1 );
    printf("Value available at **ptr2 = %d\n", **ptr2);

    return 0;
}
```

Important

1. ( ) → function
2. [ ] → array
3. * → pointer
4. Identifier → var name, fun name
5. Data type → int, char, float

1. L to R

2. R to L

3.

Examples

- int *(p[3]) → p is a array of 3 pointer to integer
- int (*p)[3] → p is a pointer to array of 3 integer
- int (*p)(int) → p is a pointer to function that takes one integer argument and return a integer value

Pointer to an array

```c
#include<stdio.h>

int main()
{
int arr[5] = { 1, 2, 3, 4, 5 };
int *ptr = arr;

printf("%p\n", ptr);
return 0;
}
```

In the above program, we have a pointer ***ptr*** that points to the **0th** element of the array. Similarly, we can also declare a pointer that can point to whole array (pointer to an array) instead of only one element of the array.

Pointer to an array

Syntax :

```
data_type (*var_name)[size_of_array];
```

Here:

- **data_type** is the type of data that the array holds.
- **var_name** is the name of the pointer variable.
- **size_of_array** is the size of the array to which the pointer will point.

Note:
The pointer that points to the 0^{th} element of array and the pointer that points to the whole array are totally different.

Example

```
int (*ptr)[10];
```

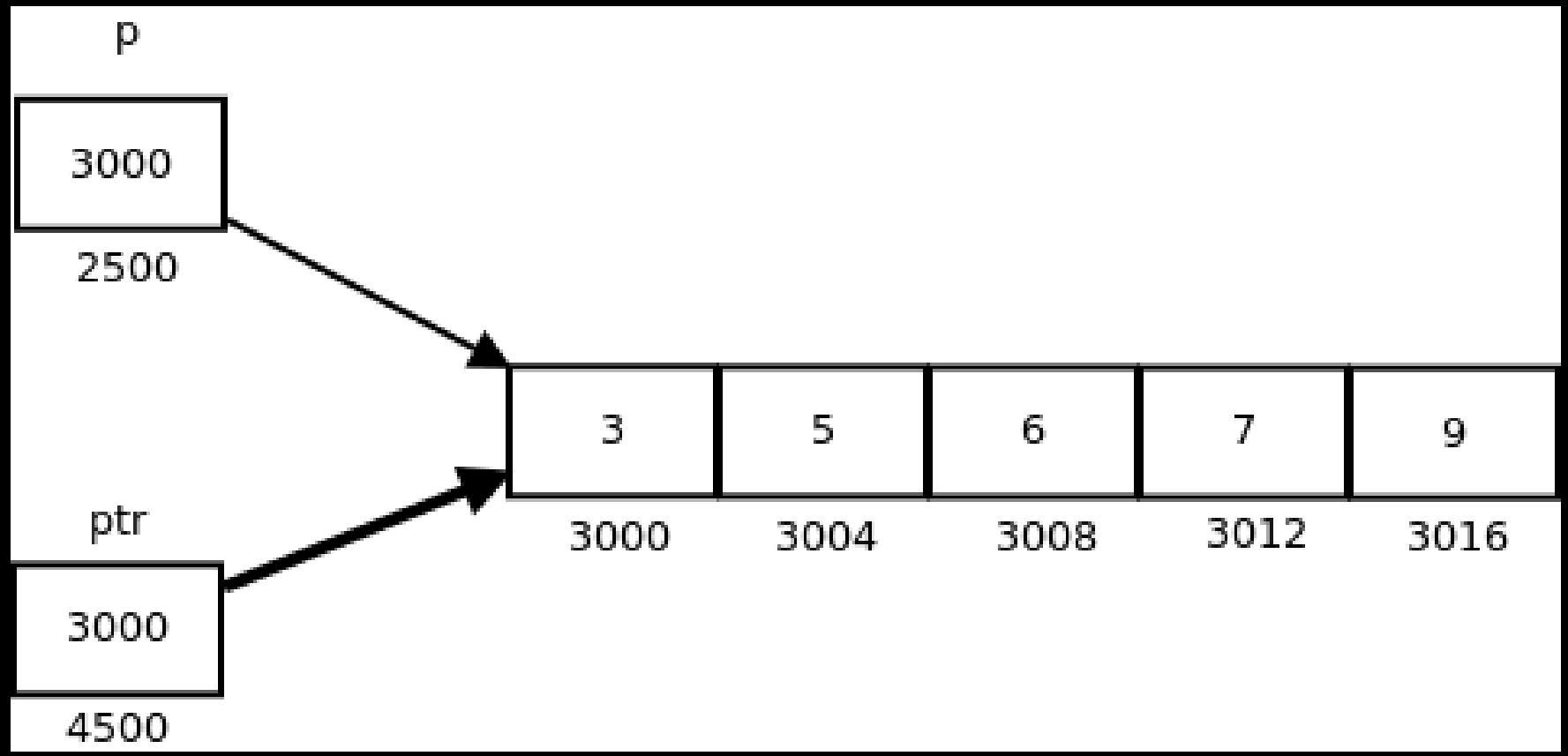
Here *ptr* is pointer that can point to an array of 10 integers. Since subscript have higher precedence than indirection, it is necessary to enclose the indirection operator and pointer name inside parentheses. Here the type of ptr is ‘pointer to an array of 10 integers.

```c
// C program to understand difference between
// pointer to an integer and pointer to an
// array of integers.
#include<stdio.h>

int main()
{
    // Pointer to an integer
    int *p;

    // Pointer to an array of 5 integers
    int (*ptr)[5];
    int arr[5];

    // Points to 0th element of the arr.
    p = arr;

    // Points to the whole array arr.
    ptr = &arr;

    printf("p = %p, ptr = %p\n", p, ptr);

    p++;
    ptr++;

    printf("p = %p, ptr = %p\n", p, ptr);

    return 0;
}
```

```
Output
p = 0x7fff6463e890, ptr = 0x7fff6463e890
p = 0x7fff6463e894, ptr = 0x7fff6463e8a4
```

Here, **p** is pointer to 0^{th} element of the array *arr*, while ***ptr*** is a pointer that points to the whole array *arr*.

- The base type of *p* is int while base type of *ptr* is 'an array of 5 integers'.
- We know that the pointer arithmetic is performed relative to the base size, so if we write ptr++, then the pointer *ptr* will be shifted forward by 20 bytes.



On dereferencing a pointer expression we get a value pointed to by that pointer expression. The pointer to an array points to an array, so on dereferencing it, we should get the array, and the name of the array denotes the base address. So whenever a pointer to an array is dereferenced, we get the base address of the array to which it points.

```c
// C program to demonstrate
// pointer to an array.

#include <stdio.h>

int main()
{

    // Pointer to an array of five numbers
    int(*a)[5];

    int b[5] = { 1, 2, 3, 4, 5 };

    int i = 0;

    // Points to the whole array b

    a = &b;

    for (i = 0; i < 5; i++)

        printf("%d\t", *(*a + i));

    return 0;
}
```

###output###

1 2 3 4 5

Array of Pointer

Pointer array is a homogeneous collection of indexed pointer variables that are references to a memory location.
It is use when we want to point at multiple memory locations of a similar data type in our program.
We can access the data by dereferencing the pointer pointing to it.

Syntax :

```
pointer_type *array_name [array_size];
```

Here,
- **pointer_type:** Type of data the pointer is pointing to.
- **array_name:** Name of the array of pointers.
- **array_size:** Size of the array of pointers.

eg

```
int *ary[55]
```

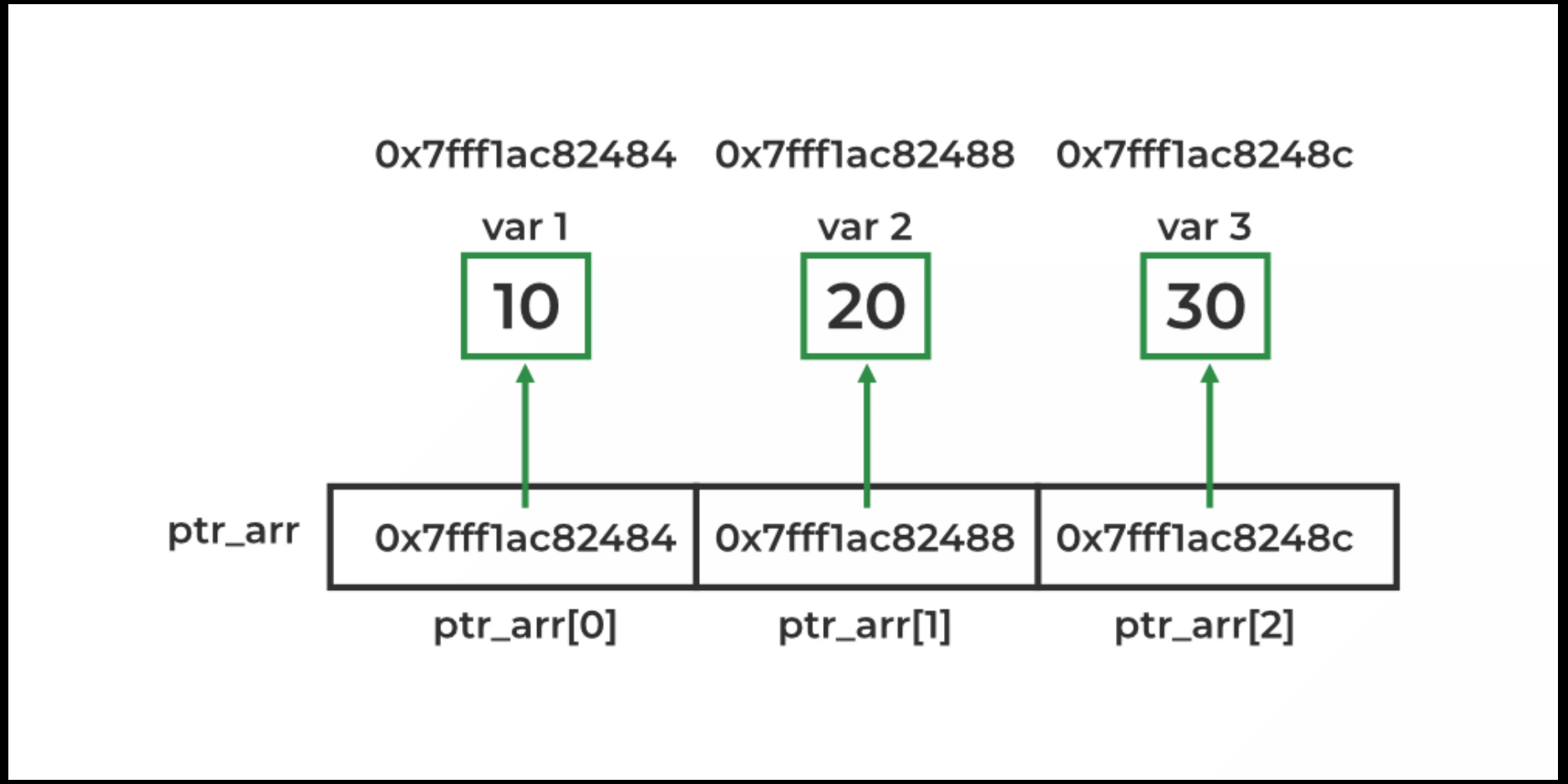
This declaration represents an array of 55 pointers.
In other words, this array can hold the addresses of 55 integer variables.
For instance,
ary[0] holds the address of one integer variable,
ary[1] holds the address of another integer variable, and so on.

```c
// C program to demonstrate the use of array of pointers
#include <stdio.h>

int main()
{
    // declaring some temp variables
    int var1 = 10;
    int var2 = 20;
    int var3 = 30;

    // array of pointers to integers
    int* ptr_arr[3] = { &var1, &var2, &var3 };

    // traversing using loop
    for (int i = 0; i < 3; i++)
    {
        printf("Value of var%d: %d\tAddress: %p\n", i + 1, *ptr_arr[i], ptr_arr[i]);
    }

    return 0;
}
```

Output

```
Value of var1: 10 Address: 0x7fff1ac82484
Value of var2: 20 Address: 0x7fff1ac82488
Value of var3: 30 Address: 0x7fff1ac8248c
```

Explanation:



As shown in the example, each element of the array is a pointer pointing to an integer. We can access the value of these integers by first selecting the array element and then dereferencing it to get the value.

Example of Using an Array of Pointers


```c
#include<stdio.h>
#define SIZE 10

int main()
{
    int *arr[3];
    int p = 40, q = 60, r = 90, i;
    arr[0] = &p;
    arr[1] = &q;
    arr[2] = &r;
    for(i = 0; i < 3; i++)
    {

        printf("For the Address = %d\t the Value would be = %d\n", arr[i], *arr[i]);
    }

return 0;
}
```

Array of Pointers to Character

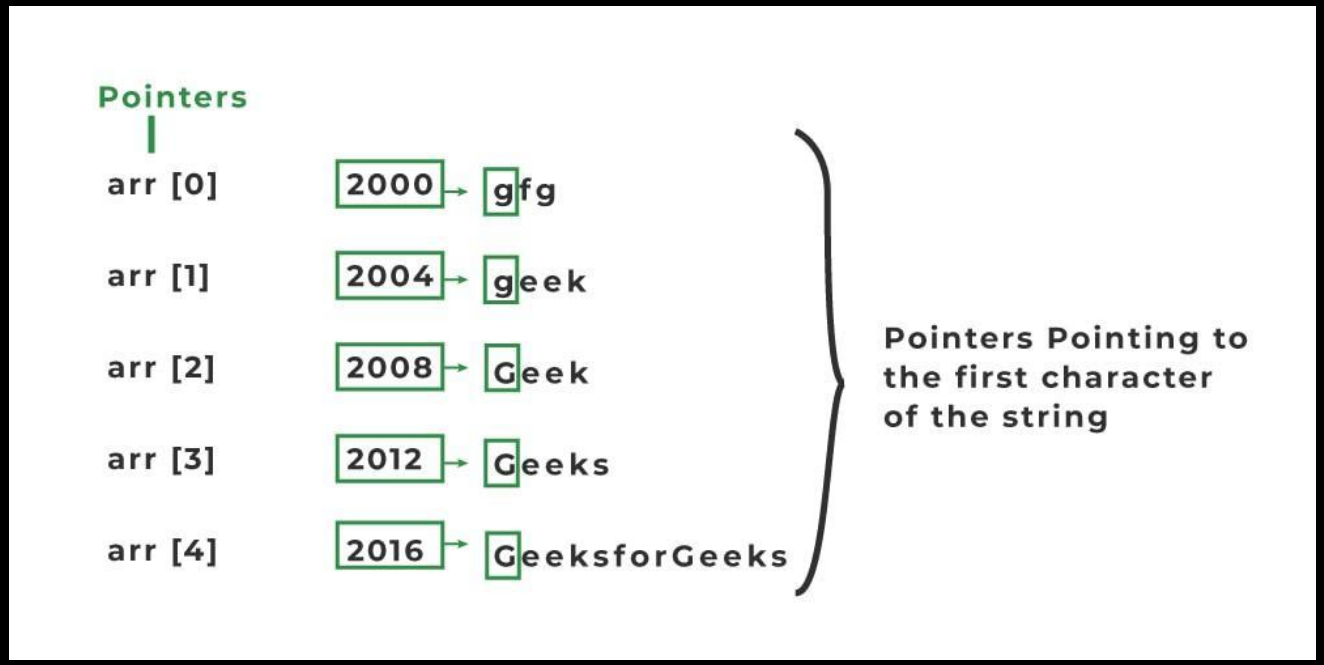
One of the main applications of the array of pointers is to store multiple strings as an array of pointers to characters.
Here, each pointer in the array is a character pointer that points to the first character of the string.

Syntax:

```
char *array_name [array_size];
```

Example

```
char* arr[5] = { "gfg", "geek", "Geek", "Geeks", "GeeksforGeeks" }
```



This method of storing strings has the advantage of the traditional array of strings.

```c
// C Program to print Array of strings without array of pointers
#include <stdio.h>
int main()
{
    char str[3][10] = { "Geek", "Geeks", "Geekfor" };

    printf("String array Elements are:\n");

    for (int i = 0; i < 3; i++) {
        printf("%s\n", str[i]);
    }

    return 0;
}
```

Output

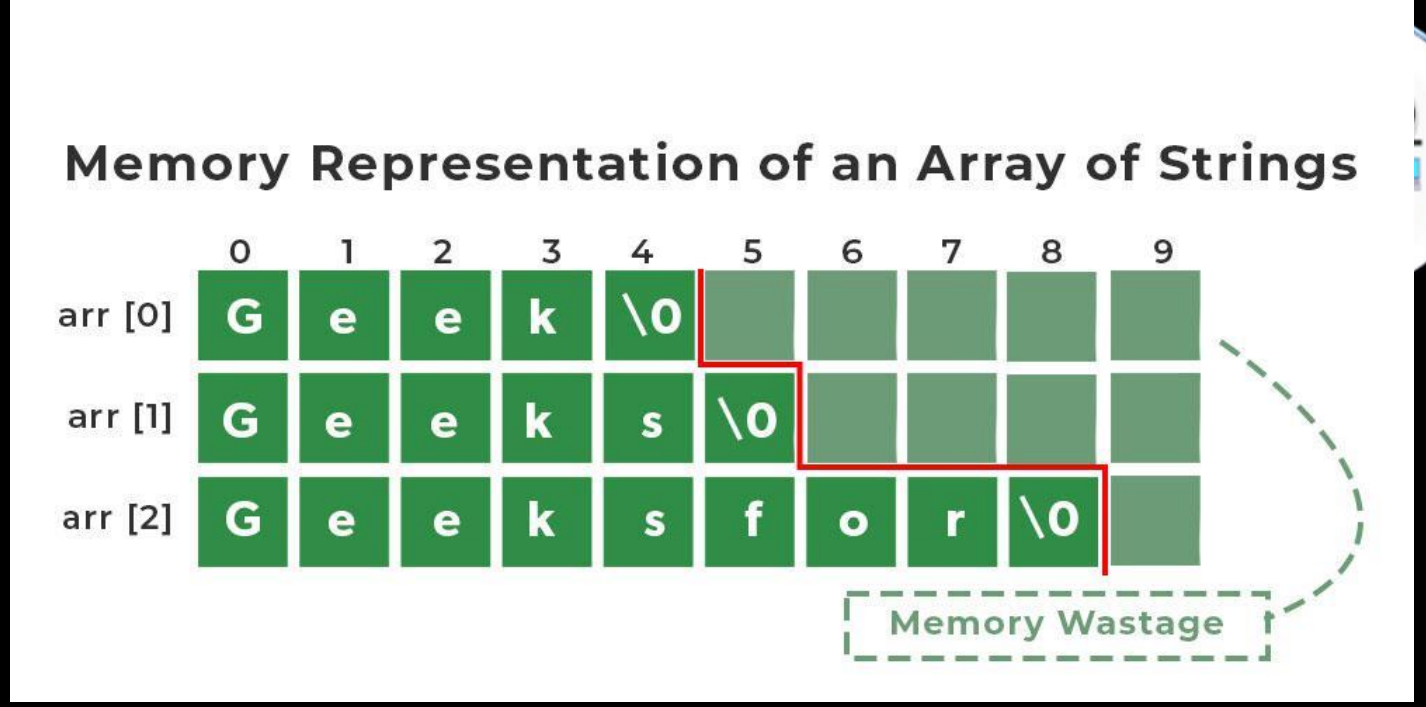
```
String array Elements are:
Geek
Geeks
Geekfor
```

In the program, we have declared the 3 rows and 10 columns of our array of strings. But because of predefining the size of the array of strings the space consumption of the program increases if the memory is not utilized properly or left unused. Now let's try to store the same strings in an array of pointers.

```c
// C program to illustrate the
//use of array of pointers to
// characters
#include <stdio.h>

int main()
{

    char* arr[3] = { "geek",
"Geeks", "Geeksfor" };

    for (int i = 0; i < 3; i++) {
        printf("%s\n", arr[i]);
    }

    return 0;
}
```

Output

```
geek
Geeks
Geeksfor
```



Here, the total memory used is the memory required for storing the strings and pointers without leaving any empty space hence, saving a lot of wasted space. We can understand this using the image shown.

The space occupied by the array of pointers to characters is shown by solid green blocks excluding the memory required for storing the pointer while the space occupied by the array of strings includes both solid and light green blocks.

Structure Pointer in C

A structure pointer is defined as the pointer which points to the address of the memory block that stores a structure known as the structure pointer. Complex data structures like Linked lists, trees, graphs, etc. are created with the help of structure pointers. The structure pointer tells the address of a structure in memory by pointing the variable to the structure variable.

```c
// structure pointer
#include <stdio.h>

struct point {
    int value;
};

int main()
{

    struct point s;

    // Initialization of the structure pointer
    struct point* ptr = &s;

    return 0;
}
```

*In the above code **s** is an instance of struct point and **ptr** is the struct pointer because it is storing the address of struct point.*

Accessing the Structure Member with the Help of Pointers

There are two ways to access the members of the structure with the help of a structure pointer:

1. With the help of (*) asterisk or indirection operator and (.) dot operator.
2. With the help of (->) Arrow operator.

Below is the program to access the structure members using the structure pointer with the help of the dot operator.


```c
//Structure pointer
#include <stdio.h>
#include <string.h>

struct Student {
    int roll_no;
    char name[30];
    char branch[40];
    int batch;
};

int main()
{

    struct Student s1;
    struct Student* ptr = &s1;

    s1.roll_no = 27;
    strcpy(s1.name, "Kks");
    strcpy(s1.branch, "Computer Science And Engineering");
    s1.batch = 2022;

    printf("Roll Number: %d\n", (*ptr).roll_no);
    printf("Name: %s\n", (*ptr).name);
    printf("Branch: %s\n", (*ptr).branch);
    printf("Batch: %d", (*ptr).batch);

    return 0;
}
```

Below is the program to access the structure members using the structure pointer with the help of the Arrow operator. In this program, we have created a Structure Student containing structure variable s. The Structure Student has roll_no, name, branch, and batch.


```c
// Structure pointer -> operator
#include <stdio.h>
#include <string.h>

struct Student {
    int roll_no;
    char name[30];
    char branch[40];
    int batch;
};

// variable of structure with pointer
defined
struct Student s, *ptr;

int main()
{

    ptr = &s;

    printf("Enter the Roll Number of Student\n");
    scanf("%d", &ptr->roll_no);
    printf("Enter Name of Student\n");
    scanf("%s", &ptr->name);
    printf("Enter Branch of Student\n");
    scanf("%s", &ptr->branch);
    printf("Enter batch of Student\n");
    scanf("%d", &ptr->batch);

    // Displaying details of the student
    printf("\nStudent details are: \n");

    printf("Roll No: %d\n", ptr->roll_no);
    printf("Name: %s\n", ptr->name);
    printf("Branch: %s\n", ptr->branch);
    printf("Batch: %d\n", ptr->batch);

    return 0;
}
```

Function Pointer in C

In C, like normal data pointers (int *, char *, etc), we can have pointers to functions. Following is a simple example that shows declaration and function call using function pointer.

```c
#include <stdio.h>
// A normal function with an int parameter
// and void return type
void fun(int a)
{
    printf("Value of a is %d\n", a);
}

int main()
{
    // fun_ptr is a pointer to function fun()
    void (*fun_ptr)(int) = &fun;

    /* The above line is equivalent of following two
    void (*fun_ptr)(int);
    fun_ptr = &fun;
    */

    // Invoking fun() using fun_ptr
    (*fun_ptr)(10);

    return 0;
}
```

```
/*
OUTPUT
Value of a is 10
*/
```

Example

1. Addition of two number
2. User input in pointer
3. Subtraction of two number
4. Multiplication of two number
5. Division of two number
6. Table of one
7. Pointer to array
8. Pointer to function
9. Pointer to structure

Advantages of Pointers in C

- Pointers in C programming are helpful to access a memory location
- Pointers are an effective way to access the array structure elements
- Pointers are used for the allocation of dynamic memory and the distribution
- Pointers are used to build complicated data structures like a linked list, graph, tree, etc

Disadvantages of Pointers?

- Pointers are a bit difficult to understand
- Pointers can cause several errors, such as segmentation errors or unrequired memory access
- If a pointer has an incorrect value, it may corrupt the memory
- Pointers may also cause memory leakage
- The pointers are relatively slower than the variables

Thank You !!

Dhanybad !!

Shukriya !!